Genetic Algorithms using Hadoop MapReduce

Rakesh Yadav
Roll. 213CS1155

under the supervision of
Dr. Ashok Kumar Turuk

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela – 769008, India

Genetic Algorithms using Hadoop MapReduce

Dissertation submitted in

the department of

Computer Science and Engineering

(Specialization: Computer Science)

of

National Institute of Technology Rourkela

in partial fulfillment of the requirements

for the degree of

Master of Technology

by

Rakesh Yadav

(Roll. 213CS1155)

under the supervision of

Dr. Ashok Kumar Turuk

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela – 769 008, India

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela - 769008, Odisha, India

CERTIFICATE

This is to affirm that the thesis entitled ***Genetic Algorithms using MapReduce*** submitted by ***Rakesh Yadav***, to the Department of Computer Science and Engineering, in partial fulfillment for the award of the degree of **Master of Technology** (**Computer Science**), is a record of factual exploration work did by him under my supervision during the period 2014-2015. The thesis has satisfied all the necessities according to the regulations of the Institute and in my opinion, has come to the standard required for submission.

Dr. Ashok Kumar Turuk
Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769008, Odisha, INDIA

Place: NIT Rourkela
Date: May 25, 2015

Dedicated to my Parents and Siblings

Acknowledgment

First of all, I like to thank Almighty God, who has blessed me so that I am able to accomplish this thesis as a partial fulfillment of the requirement for the degree of Master of Technology in Computer Science and Engineering.

I would like to express the deepest gratitude towards my supervisor Dr. Ashok Kumar Turuk for pushing me to work in the area of Genetic Algorithm using MapReduce. I am sincerely thankful to him for giving his valuable time, advice and correction to the thesis from the beginning till the end. I also wanted to thank all the faculty members of Computer Science Department for their genuine suggestion, advice and corrections. Without their guidance and encouragement it would not be possible for me to complete this thesis.

Further, I would like to express my sincere thanks to all my classmates and PhD Scholars, especially R.S. Barpanda for sharing his knowledge at the initial stage of my research work.

I must acknowledge the alumina for providing needful resources and platform which are the milestone in enhancing my knowledge and skillfulness required to write this thesis.

Finally, at this moment I surrender everything what I have is an outcome of the support and constant believe of my beloved parents. I am dedicating my thesis to my family especially to my mother, Lal Dei Yadav for always standing next to me irrespective of her own compromises and my father, R.S. Yadav for his encouragement and support.

Rakesh Yadav

Abstract

Data-Intensive Computing (DIC) played an important role for large data set utilizing the parallel computing. DIC model shown that can process large amount of data like petabyte or zettabyte day to day. So these have some sorts of attempts to checkout that how DIC will support the Evolutionary(Genetic) Algorithms. Here we have shown step by step explanation that how the Genetic Algorithms(GA), with different implementation form, will be interpret into Hadoop MapReduce framework. Here the results will give details as how Hadoop is best choice to impel genetic algorithm on large dataset problem and shown how the speedup will be increased using parallel computing.

MapReduce is designed for large volume of data set. It is introduced for BigData Analysis and it is used a lots of algorithms like Breadth-First Search, Traveling Salesman problems, Finding Shortest Path problem etc. In this framework two key factor, Map and reducer. The Map which is parallely divided the data into many cluster and each cluster the data is form of key and value. The output of map phase data will goes into intermediate phase where data will be shuffling and sorting. Then using the partitioner for dividing the data parallely in different cluster according to the user. The number of cluster are depends on the number of reducers. The reducers will taking all iteration of data give the results in form of values.

In This thesis we also show that we compare our implementation with implementation presented in existing model. These two implementation are compare with ONEMAX (bit counting) PROBLEM. The comparison criteria between two implementation are fitness convergences, stability with fixed number of node, quality of final solution, cloud resource utilization and algorithms scalability.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Every day in life data may increasing exponentially in different domains of computer application and Internet world. Due to increasing large data we are forcing to rethinking how to processing the large volume of data. The most data-intensive computing frameworks [3] [4] show some common characteristic like : data-flow processing [5]. Availability of dataset is not only processing but also show the behaviors of distributed data execution. In this era the research and other communication process have a lot of data information, Which are processed into data intensive framework [6]. There are many research are done in the field of parallelizing computation algorithms [7, 8] but little work are done in the field of data intensive computing [9]..

The intrinsic parallel behaviors of GA are make them alternatives input for distribution. Hence focus in this thesis, GA and its intrinsic require to work with BigData volume, improvident if they take the form of populations of individuals or specimen out of a probability of distribution, could significantly advantages from a DIC modeling. this thesis, we are focusing the use Hadoop model introduced by Yahoo [10] and execution of MapReduce.

1.2 Motivation

The data intensive [9] is large data set. MapReduce perform on a input set in the order of petabytes in size. So these are more significance to the operation on MapReduce to be able to handle and run on data that may not entirely fit into the cluster's memory. And the data are different locations but mapreduce [1] easily process on the data.

The related parallelized implementation of big dataset execution are simply perform. In every map phase are separate to each other and they are perform the re-execution which provide fault tolerance mechanism.

An evolutionary (Genetic Algorithms) computation community [5] of embracing there proposals, They are choosing various illustrative algorithms and progressed their respective MapReduce implementations. These are useful to remember that we focus more awareness to assurance that underlying process was not changed and the behaviors are maintained of algorithms.

1.3 Objective of our thesis:

- To propose an algorithm that can reduce the time of process for large volume of dataset.
- To propose an algorithm that can useful for fault tolerance and data availability.
- the parallelized MapReduce GA will reduce the time of process and give the result efficiently.
- This will be useful for big data analysis.
- To propose an algorithm that generate the different number of data set and each of them will work parallel way.

- To propose an algorithm that Hadoop uses the HDFS for storing the large amount of data and process then independently.

1.4 Organization of the Thesis

The rest of the thesis is organized as follows:

1. Chapter 1: In this chapter we have discussed about the introduction to thesis, motivation and objective of our research.

2. Chapter 2: In this chapter we present the basic concepts and definitions of Hadoop , architecture of hadoop , MapReduce and HDFS overview etc.

3. Chapter 3: In this chapter we are discuss about the genetic algorithms and its variants names as simple genetic algorithms and compact genetic algorithms.

4. Chapter 4: In this chapter we are discuss about MapReducing genetic algorithms and how inherited genetic algorithms into mapreduce framework

5. Chapter 5: In this chapter we have proposed the MapReduce genetic algorithms and compare with existing algorithms

6. Chapter 6: At last we concluded our work and discuss few future works possible on this area.

Chapter 2

Basic Concepts and Defination Hadoop MapReduce

2.1 Introduction of Hadoop

Hadoop [10] is the Apache open source framework indited in Java which sanctions distributed processing of astronomically immense datasets across collection of computers system utilizing programming framework. The Hadoop framework application works in an environment that provides storage in distributed manners and computation across clusters of computers. Hadoop is setup to the such a manner that single server have a lot of machines working and each will provide local computation process and storage system.Those are some costly to create larger servers with more efficient configurations that will process large scale data, but as another way, we could add together many cluster of computers with single-CPU, as a single functional distributed system.The clustered machines could read the dataset in parallel and give a more higher throughput results [3]. So, these are cheaper than one high-end server. So this is the first motivational factor behind using Hadoop that run across clustered and low-cost machines. Hadoop run its code from different cluster of machines and perform the following task in process.

- The data is divided into many number of records and directories. the records are uniformly divided into 128M and 64M size of blocks.
- These blocks of files are distributed into various nodes to processing the data set.
- The HDFS [11] are used in the top of local file system to handle the processing.
- The cluster of blocks are replicated to control the hardware failure so this will provides the fault tolerance properties.
- In hadoop checking the code war performing successfully or not.
- For performing the sort of action that will put between the map and reduce phases.
- After performing the mapreducing the sorted data will be sending to the computer node.
- after this witting the debugging logs for every jobs.

Advantages of Hadoop:

- Hadoop framework will grant the user to writing and examining distributed systems [12]. These are efficient, and it will automatic distributes the dataset and perform across the various nodes and in turn, utilizes the elementary parallelism of the CPU cores [4].
- Hadoop is not dependent on the hardware for providing the fault tolerance behavior and high availability, rather the library function of hadoop have designed in such a manner to find and improve the faults at the application layers.
- The servers are adding or removing from the cluster dynamically and operation are performed without interruption.

- One of big advantage of Hadoop is product of Apache Software Foundation which is open source . This is compatible on all the platforms since it is Java based.

2.2 Architecture of Hadoop

In the architecture of hadoop [10] we describes the various components of hadoop like mapreduce job processing, handling the data and architecture of file system. Hadoop deploys a master/slave architecture for distributed computation and distributed storage [13].

The various component of Hadoop :

1.Processing / computation layer (MapReduce)

2. Storage layer(Hadoop Distributed File System) [11]

3. Hadoop comman

4. Hadoop YARN

Hadoop comman : Hadoop comman are Java libraries and utilities which is require by another hadoop modules.

Hadoop YARN : Hadoop YARN is the framework using for job scheduling and system resource management.As clusters grew and adoption expanded, And there are many ways to intact the user with data store in Hadoop. In the prosperous projects there are many tools are used to interact with data source like Apache Hive are used for SQL-predicted query , Apache Pig is used for processing the scripted data and Apache HBase is used for NoSQL database [14].

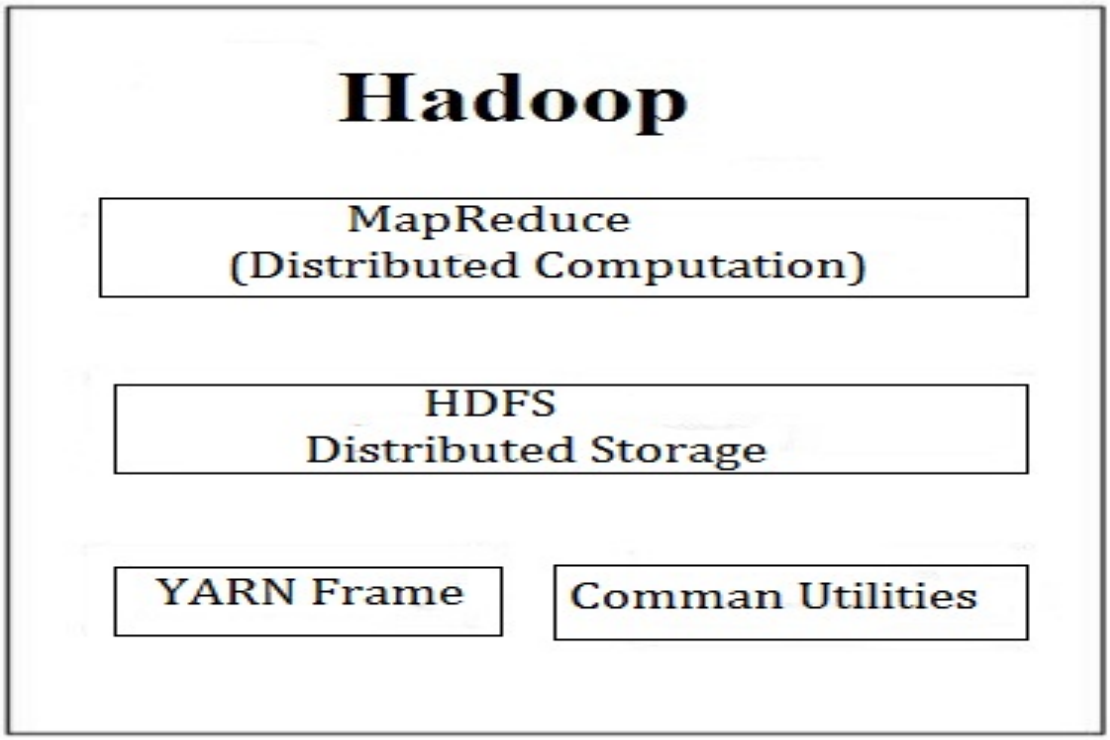

Figure 2.1: Hadoop Architecture

2.3 MapReduce

MapReduce framework are proposed by Google, used to write application to processing large dataset , collateral way , on more cluster of hardware system in efficient way [15].

MapReduce is processing technique and programming model for distributed computing based on JAVA programming language. The MapReduce algorithms have two component namely Map and Reduce.Map takes the large volume of dataset and convert them into broken form of tuples as (key/value) pairs and Reduce phase takes the output of maps phases and combine them into smaller tuples sets. According to name sequence Reduce task will perform after the Map phase operations.

MapReduce framework is designed by Google by seeing the behaviors of map reduce functional programming languages. MapReduce is enables to easily use for

large distributes problems. In MapReduce framework there are two components, one is map and other is reduce.

MapReduce is designed for large volume of data set. It is introduced for BigData Analysis [16] and it is used a lots of algorithms like Breadth-First Search, Traveling Salesman problems, Finding Shortest Path problem etc. In this framework two key factor, Map and reducer. The Map which is parallely divided the data into many cluster and each cluster the data is form of key and value. The output of map phase data will goes into intermediate phase where data will be shuffling and sorting. Then using the partitioner for dividing the data parallel in different cluster according to the user. A cluster are determined by the number of reducers. The reducers will taking all iteration of data give the results in form of values.

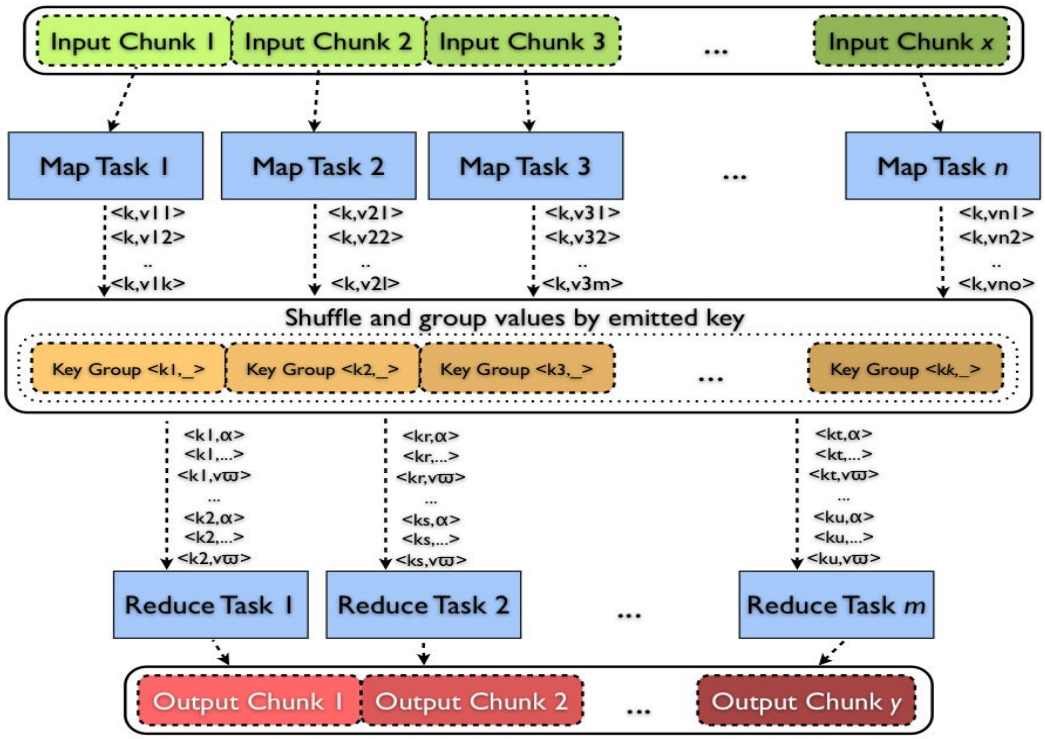

Figure 2.2: MapReduce framework [1]

The advantages of MapReduce is that it easily scale the processing the dataset on multiple system nodes. In the MapReducer model processing primitives of

data is called mapper and reducer. mappers and reducers decomposition of data processing are mostly nontrivial but we indite the MapReduce application form, comparing the application to execute on thousands cluster of machines is only change in configuration. The scalability has magnetized many programmers to utilize the MapReduce model in the effective manner.

The Algorithms of MapReduce:

The MapReduce Program are excuted into the three number of phases, as Map phases , shuffle phase and Reduce phase.

- **Map phase :** In this phase the mapper's job will be processed as the input dataset and generally the dataset are in form of file or directory and store in locally Hadoop File System(HDFS).And the input dataset are passed into mapper in line by line and mapper creates the many small chunk of data.
- **Shuffle phase :** In the shuffle phase the input data which comes from mapper is shuffling and sorting the the various key/value dataset and gives the new dataset.
- **Reduce phase :** The Reducer phase process the data and produce the new output and store the HDFS.
- **Inputs and Outputs of MapReduce :** The MapReduce will work on the ¡key,value¿ pair dataset.

	Input	Output
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

Figure 2.3: Input and Output of MapReduce

Terminology :

- **Payload -** It is an application which implement Map and reduce function and useful in form of core job.
- **Mapper -** It will maps the inputs key/value data into the intermediate data sets.
- **NameNode -** The NameNode have information about the every nodes cluster.These are useful for Hadoop Distributed File System(HDFS).
- **DataNode -** DataNode is the node or system where the data will be presented already before processing will happens.
- **MasterNode -** MasterNode is the node where the JobTracker will perform and takes the request of job from clients.
- **SlaveNode-** This will be the node where every mapreduce program will be executed.
- **JobTracker-** It will be scheduling the job and it will track to the TaskTracker that that will working or not.
- **TaskTracker-** this will track the various task and inform to JobTracker.

2.4 HDFS Overview

Hadoop can work directly with any mountable distributed file system, but the most mundane file system utilized by Hadoop is the Hadoop Distributed File System (HDFS) [11]. It is a fault-tolerant distributed file system that is designed for commonly available hardware. It is well-suited for astronomically immense data sets due to its high throughput access to application data. In the HDFS architecture there are many content are available like NameNode and DataNode. The NameNode have information of every DataNode and every execution are performed in the DataNode.

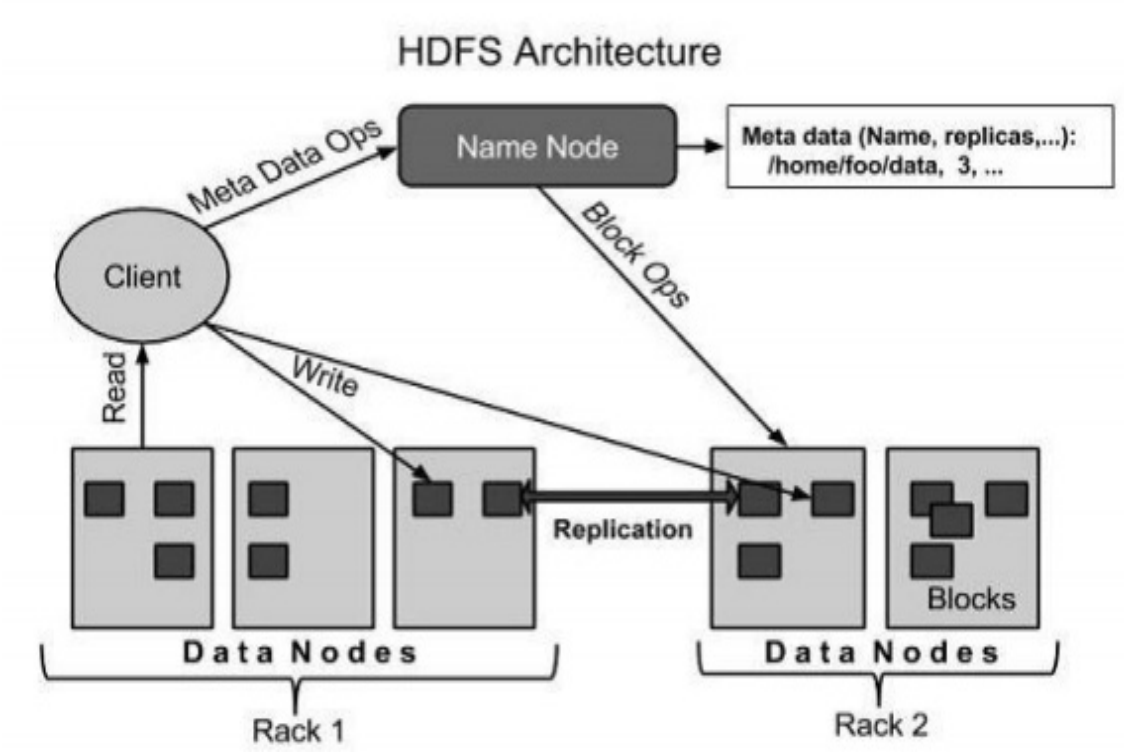

Figure 2.4: Architecture of HDFS

Chapter 3

Genetic Algorithms (GAs)

The Genetic algorithms [17] is different from the other search algorithms.In the GAs there are many sample solution and one of sample solution is candidate solution or partial solutions. Every sample solutions of problem in the data structure is known as an individuals. The individuals have two component.First is chromosome and second is fitness. The chromosome is the bunch of genes.The collection of the chromosome is called as populations. In genetic algorithms , at the time of search process the size of populations is remains constants. The current population selection is known as parents and selection are performed on the basis of fitness values and allow them to create offspring.

In the GAs, We see that the sample which have above average fitness have more chance to selected in the GAs. After the selection of the fitness the reproduction process such as crossover and mutation are applied on sample solutions. After the crossover the parents copies their genes and generate new chromosome for the offspring.The mutation process have needed only one parents. An offspring are created only resembles the only few number of genes and created the children and the children also getting the fitness. When the children are included in populations, Some of individuals populations have to die and the children takes those places. Basically the removing task are performed on the basis of fitness values which have below average individuals have more chance of selection for die.

The procedure of selecting the individuals for generate new or dead is depends on his relatives fitness are known as natural selection. The genetic algorithms are totally based on the selection and crossover.

The procedure of assigning individuals to new generation or dead depend on on his relative fitness are known as natural selection. Every particular that is best fit is allowed to survive more and regenerate most often. The useful points of GAs is that the starting population need not better of individuals.

Initially the sample solutions are generated randomly and selected one of the is candidate solutions. But using the selection and reproduction many times the genetic algorithms get the satisfactory solution very efficiently and quickly.

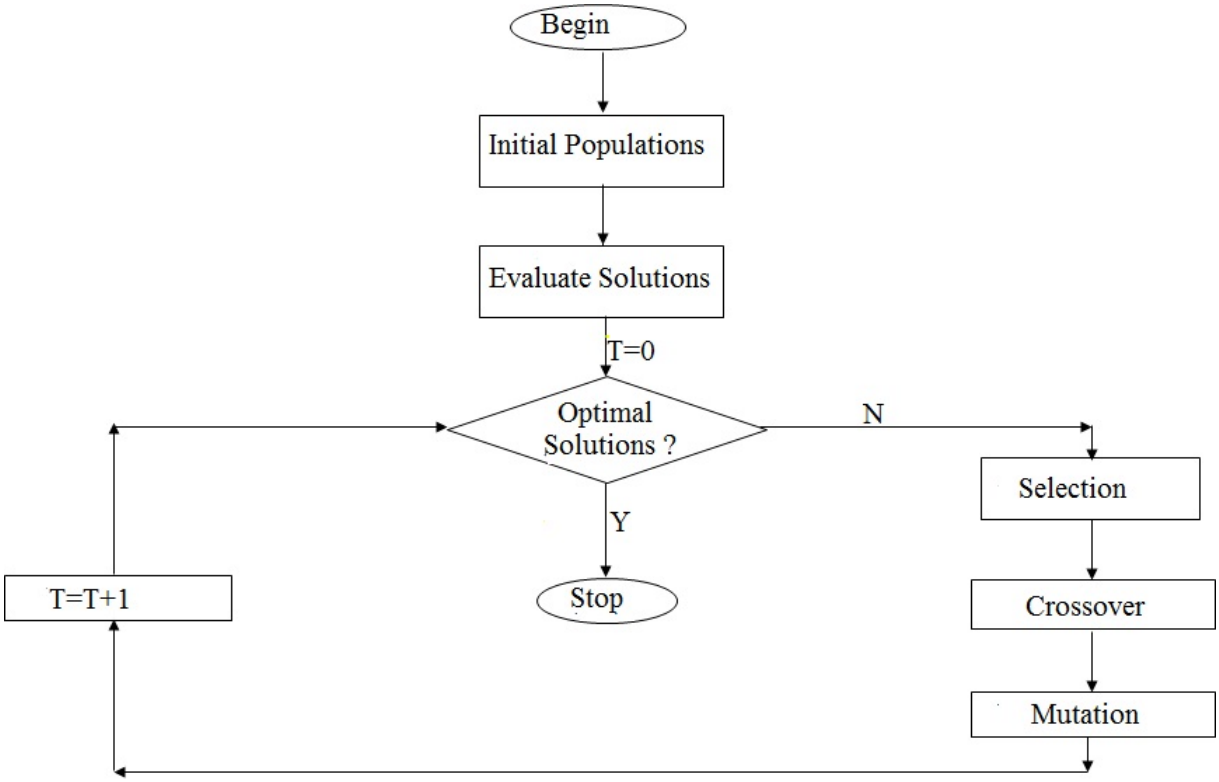

Figure 3.1: Mechanism of GAs

There are various variants of Genetic algorithms :

1. Simple Genetic Algorithms
2. Compact Genetic Algorithms

3.1 Simple Genetic Algorithms(SGAs)

The Recombination GAs [18], is simplest forms of GAs. It is mainly using for the selection and recombination. We choose the initials populations with random individuals and find fitness values of every individuals populations. The main objective of using the Simple Genetic Algorithms is to implement the data-intensive flow as following:

1. Selecting initials populations as randomly.

2.Calculate the fitness of each individuals.

3.Choosing best results applying s-wise tournament selection [19] process without replacement [16].where s is random individuals choosing from populations.

4.Generate new individuals values and recombining the choose population applying uniform crossover.

5.Again calculating the fitness of new generated individuals.

6. iterate 3-5 step until the criteria of convergence is satisfied.

3.2 The Compact Genetic Algorithm(CGAs)

CGAs [20] is a simple estimation distribution algorithm. It builds probabilistic models in producing candidate solutions by using model specimens. This method assumes that each gene is free and has nothing in common as well there exist no interdependency among them. The population is used to be represented using the probability vector which further assists search by producing candidate solutions.
The CGA comprises of the below mentioned steps:

- **1. Initialization:** The population in simple genetic algorithm is initialized with some random individuals. The compact genetic algorithm starts with probability vector which sets to 0.5 initially.
- **2.Model sampling:** Probability vectors are used to obtain the candidate

results. These procedure are same as the uniform crossover in the genetic algorithm.

- **3. Evaluation :** Calculating the fitness paradigm and quality assurance is carried out.
- **4. Selection:** Compact genetic algorithm apply selection process over the available individuals to find out the best among them.
- **5. Probabilistic model update:** The winning proportion is increased by 1/n on completion of selection.

$$Q_{yi}^{m+1} = \begin{cases} Q_{yi}^{m} + \frac{1}{y} \text{ If } y_{w,i} \neq y_{c,i} \text{ and } y_{w,i} = 1, \\ Q_{yi}^{m} - \frac{1}{y} \text{ If } y_{w,i} \neq y_{c,i} \text{ and } y_{w,i} = 0, \\ Q_{yi} \text{ otherwise} \end{cases} \tag{3.1}$$

Here, $y_{w,i}$ is the i^{th} chromosome winning gene, $y_{c,i}$ are i^{th} gene of competing chromosome, andQ_{Yi}^{m} is thei^{th} components of the vector probability showing the portion of i^{th} gene is become one of generation m. The procedure of update compact genetic algorithms is similar attributes of the GA having population size of m and using binary tournament selection [19].

- **6.** Iterate steps $2-5$ unless some finishing step encountered.

The CGAs models are equivalent to the models used in PBIL (population based incremental learning) [21] and to the UMDA (univariate marginal distribution algorithm) [22]. CGA could reproduce a GAs having size of population. So it can also alter the probability vector which can't be achieved using PBIL and UMDA. Also CGA increases the memory utilization as compared to the PBIL and simple

genetic algorithm. Unlike simple genetic algorithm CGA only needs to store a small portion of n bits that can be stored in log n. It has been significantly proved that compact genetic algorithm is functionally similar to the order-one behavior. So the estimate of the parameters and CGAs behavior can be conducted using the theory of simple genetic algorithm directly.

Chapter 4

MapReducing Genetic Algorithms

In this Part [2], we are representing the two models. In the First model [18] which are using one MapReduce function for one generation of GAs and next generation uses another MapReduce phase. So the one populations for all the node or mapper in this algorithms. In the second model which are probability based, in this model one MapReduce phase uses for all the generations of GAs.

4.1 MapReducing Simple Genetic algorithms(SGAs)

In the MapReducing SGAs we will use the every iteration of genetic algorithms as a input of MapReducing phases.The simple genetic algorithms are based on the simple selection and recombination based process. We chose this algorithms because this will uses minimum number of operator that is show the data-intensive process.

4.1.1 Map

In Map function calculating [2] the fitness value of the initials values (From 2 to 5 step) which are matching with the MAP function. As shown in this Algorithm first, That the MAP phase calculating the best fitness by using initials values and find

the best fitness values. The best fitness values are storing file in globally on Hadoop Distributed File System (HDFS).

Algorithm 1 Map phase of each iteration of the GA

```
1: MAP(key, value):
2: individual ← INDIVIDUALREPRESENTATION(key)
3: fitness ← CALCULATEFITNESS(individual)
4: EMIT (individual, fitness)

5: {Keep track of the current best}
6: if fitness > max then
7:         max ← fitness
8:         maxInd ← individual
9: end if

10: if all individuals have been processed then
11:         Write best individual to global file in DFS
12: end if
```

Figure 4.1: Map phase SGAs

4.1.2 Partitioner

In the selection process of the GAs in step3 is performed locally on every node, so this process is reducing the time of the selection and it will be growing in the convergence time. So parallelized selection algorithms [23] is used. In this MapReduce framework at there is a Distributed File System use in globally for communication for the shuffle between the Mapper and Reducer. The Map Model, this model are shuffling the key/value pairs passes in the reducing applying these partitioner. So this practitioner divides the intermediate key/value pairs among the different reducers. The GETPARTITION () function gives to the reducer for which the resulting output (key, value) would be sent to. The usual execution of, user apply hash function h(k)%N in this equation N is number of reducer. The value corresponding to key will manage in corresponding reducer.

Algorithm 2 Random partitioner for GA

```
1: int GETPARTITION(key, value, numReducers):
2: return RANDOMINT(0, numReducers - 1)
```

Figure 4.2: Partition phase SGAs

4.1.3 Reduce

In the Reduce phase [2] we are using Tournament selection without replacement [19]. The tournament where we are choosing the winner among the many population according to the values. This actions are iterated population many a times. Hence any a ways choosing individuals are identical to stumbling all map set data and after this sorting those intermediate data, the reduce task process along with the every individuals consecutively.

At the beginning the initials were buffering with the final stage. The tournament window array is full, then the SELECTIONANDCROSSOVER function are completed as mention the Algorithm third. When the crossover window is full, then we are using a Uniform Crossover function. Due to execution time , we set the S to 5 and crossover are executed using two successively choose parents. The selection and crossover are executed on the data of mapper fitness values and they will gives the results of individuals and dummy fitness.

The reduce phase the number of output decided by the user because number of partitioned decided the number of reducer.

The Reducer will work on the tournament selection and check for maximum values then using the selection and crossover function.

4.1.4 Optimizations

In previous experiments we see due to big larger problem size, The serial initialization population will take more time for processing. Amdahl law theory the speed-up factor are totally depends on the serial components.

Algorithm 3 Reduce phase of each iteration of the GA

```
1: Initialize processed ← 0, tournArray [2· tSize], crossArray [cSize]

2: REDUCE(key, values):
3: while values.hasNext() do
4:         individual ← INDIVIDUALREPRESENTATION(key)
5:         fitness ← values.getValue()

6:         if processed < tSize then
7:                 {Wait for individuals to join in the tournament and put them for the last rounds}
8:                 tournArray [tSize + processed%tSize] ← individual
9:         else
10:                {Conduct tournament over past window}
11:                SELECTIONANDCROSSOVER()
12:        end if
13:        processed ← processed + 1

14:        if all individuals have been processed then
15:                {Cleanup for the last tournament windows}
16:                for k ←1 to tSize do
17:                        SELECTIONANDCROSSOVER()
18:                        processed ← processed + 1
19:                end for
20:        end if
21: end while

22: SELECTIONANDCROSSOVER:
23: crossArray[processed%cSize] ← TOURN(tournArray)
24: if (processed - tSize) % cSize = cSize - 1 then
25:        newIndividuals ← CROSSOVER(crossArray)
26:        for individual in newIndividuals do
27:                EMIT (individual, dummyFitness)
28:        end for
29: end if
```

Figure 4.3: Reduce phase SGAs

according to Amdahl law if s is the proportional of parallel program and (1-s) is non-parallelized program then maximum speedup factor of k processor are achieved in limits , as k leads to infinity leads to 1/(1 - s). So speedup depends on serial components.

hence here in this MapReducing SGAs taking initial populations as septate MapReducing job. Where Map Generate random starting populations and reduce will identity reducer.here produce the pseudo-random number generator for every mapper with mapper_id.current_time. The variables size bits in the individual are shown in form of array long long int and do for effectively bit counting actions for calculating crossover and fitness. For the incapability of representing loops in the MapReduce framework, each iteration residing of a Map and Reduce, and this will executed until the criteria of convergence are fulfilled.

4.1.5 Results of SGAs Using ONEMAX (BitCount) Problem

For showing the results we are using the one max problems.

ONEMAX (BitCount) Problem : The ONEMAX Problem [24] is the problem in which minimize the number of one in the bit-strings. The bit-string is shown as

y = {y1, y2, . . . , yN}, with yiϵ(0,1), that maximizes the following equation:

$$F(\vec{y}) = \sum_{i=1}^{M} y_i \tag{4.1}$$

Use this problem and perform the following experiments.

Genetic algorithm convergence with constant number of MapReduce tasks : Here we will using the best fitness values for testing both of model with fix the probability of crossover and mutation for fixed number of population size.

In this model various genetic algorithm contents are following:

1. Prob(Crossover)= 0.6
2. Prob(Mutation) = 0.01

3. Total population = 10000
4. Number of iteration =350

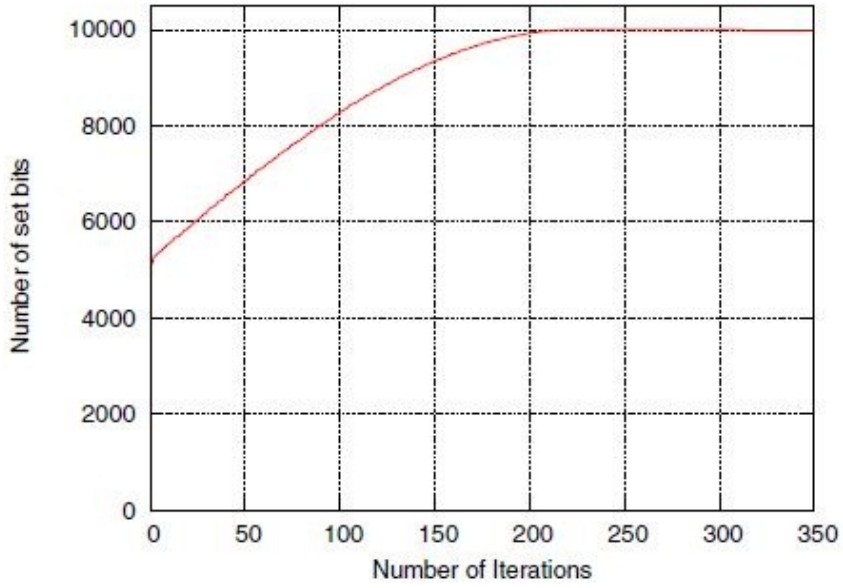

Figure 4.4: Convergence of GA for 10^4 variable OneMAX problem

Scalability with constant load per node : Here , we have take the load set upto the 15,00 variables per mapper. The time per iteration will increases first and after fixed around 76 sec . So, increasing the size of problem as many resources have joined and they are not change the time of iteration.

4.2 MapReducing Compact Genetic Algorithms(CGAs)

In this section confine every iteration of the compact Genetic Algorithms [2] as a distinct single MapReduce job. The user takes the command-line parameters, generate starting probability vector splits and submits the MapReduce job. Let the

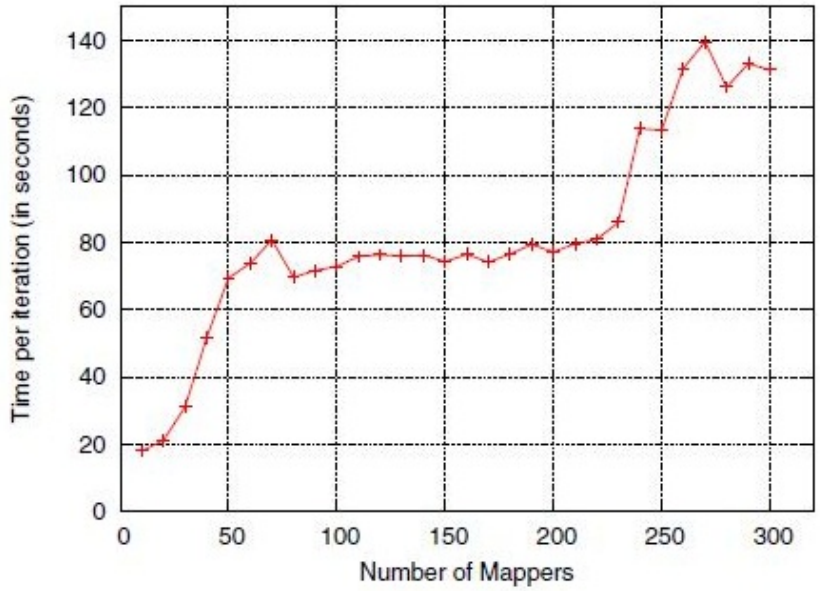

Figure 4.5: OneMAX problem with constant load per node Scalability for GAs

probability vector be P = pi : pi = the variable Probability(i) = 1. So this process should permit us to scaling in terms of the number of variables, when P is divided into k different partitions P1, P2, . . . , Pk where k is the no. of mappers.

4.2.1 Mapper :

In the Map Phase more number of individualsgeneretion are executed independently at any instances. In the below algorithms the Map calculate split probability p_i as input and output of tournamentSize individuals as well as probability spits. The Map take data to update information in Hadoop Distributed File System(HDFS). And when needed Reducer will take the data form HDFS.

Algorithm 4 Map phase of each iteration of the CGA

```
1: MAP(key, value):
2: splitNo ← key
3: probSplitArray ← value
4: EMIT(splitNo, [0, probSplitArray])
5: for k ←1 to tournamentSize do
6:         SELECTIONANDCROSSOVER()
7:         processed ← processed + 1
8:         individual ←
9:         ones ← 0
10:        for prob in probSplitArray do
11:                if RANDOM(0,1) < prob then
12:                        individual ← 1
13:                        ones ← ones + 1
14:                else
15:                        individual ← 0
16:                end if
17:                EMIT(splitNo, [k, individual])
18:                WRITETODFS(k, ones)
19:        end for
20: end for
```

Figure 4.6: Map Algorithms for CGAs [2]

4.2.2 Reducer :

Here we execute tournament pick the lack of any replacement. And tournament are manage between tournamentSize and generated individuals and after that we find the winner and looser. Then probability vector split will be updated sequentially.

Algorithm 5 Reduce phase of each iteration of the CGA

```
1: INITIALIZE:
2: ALLOCATEANDINITIALIZE(OnesArray[tournamentSize])
3: winner ← -1
4: loser ← -1
5: processed ← 0
6: n ← 0
7: for k ← 1 to tournamentSize do
8:         for r ← 1 to numReducers do
9:                 Ones[k] ← Ones[k] + READFROMDFS(r, k)
10:                if Ones[k] > winner then
11:                        winnerIndex ← k
12:                else
13:                        if Ones[k] < loser then
14:                                loserIndex ← k
15:                        end if
16:                end if
17:        end for
18: end for

19: REDUCE(key, values):
20: while values.hasNext() do
21:        splitNo ← key
22:        value[processed] ← values.getValue()
23:        processed ← processed + 1
24: end while
25: for prob in value[0] do
26:        if value[winner].bit[n] ≠ value[winner][n] then
27:                if value[winner].bit[n] = 1 then
28:                        newProbSplit [n] ← value[0] + 1/population
29:                else
30:                        newProbSplit [n] ← value[0] - 1/population
31:                end if
32:        end if
33:        EMIT(splitNo, [0, newProbSplit])
34: end for
```

Figure 4.7: Reduce Algorithms for CGAs [2]

4.2.3 Optimizations

In this MapReducing SGAs taking initial populations as septate MapReducing job. Where Map Generate random starting populations and reduce will identity reducer.here produce the pseudo-random number generator for every mapper with mapper_id.current_time. The variables size bits in the individual are shown in form of array long long int and do for effectively bit counting actions for calculating crossover and fitness. For the incapability of representing loops in the MapReduce framework, each iteration residing of a Map and Reduce, and this will executed until the criteria of convergence are satisfied.

Use this problem and perform the following experiments.

Genetic algorithm convergence with constant number of MapReduce tasks : Here we will using the best fitness values for testing both of model with fix the probability of crossover and mutation for fixed number of population size.
In this model various genetic algorithm contents are following:

1. Prob(Crossover)= 0.6
2. Prob(Mutation) = 0.01
3. Total population = 10000

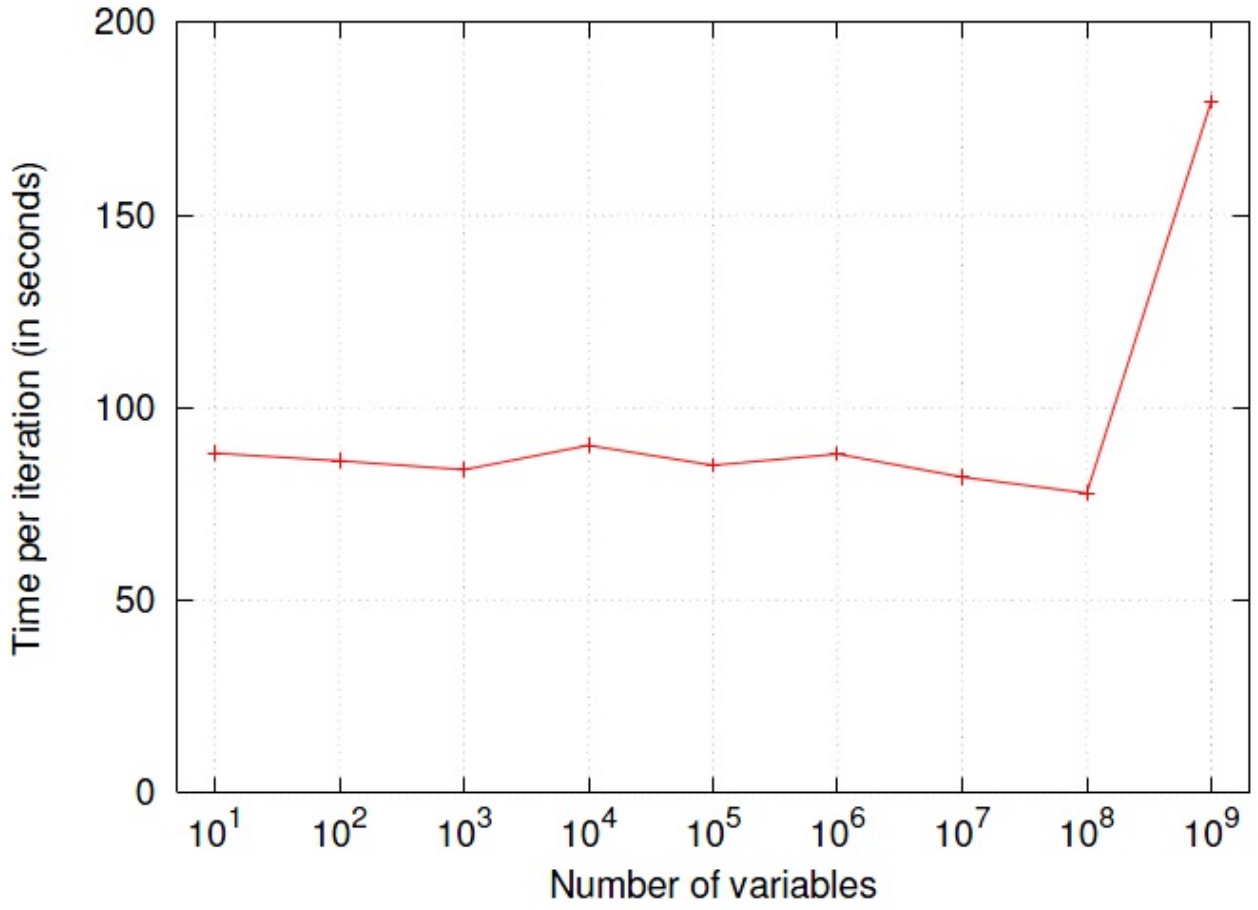

Figure 4.8: Compact genetic algorithm scalability of OneMAX problem when the variable are increasing .

Chapter 5

Proposed MapReducing Genetic Algorithms

The proposed model have the most of action moved from reduce phase to map phase. This action decreases the time of IO process because all action data is kept in system memory instead of HDFS.

In this proposed model we are using the MapReducing parallelizing Genetic algorithms in which we find scalability and convergence for given parameter.

5.1 Algorithms

In the Proposed Algorithms the every GA steps are performed in the map phase and only best result are send on the reducer phase. This action decreases the time of IO process because all action data is kept in system memory instead of HDFS.

Algorithm 1 For input phase of GA

1: popSize= define populationSize()

2: numberOfmaps= define numberOfmaps()

Algorithm 2 For map phase of GAs

```
1: currentGen=0
2: population p= generateRandomPopulation(pSize)
3: evaluateFitness(p)
4: while currentGen ¡ lastGen do
5:     p=nextGen(p)
6:     evaluateFitness(p)
7:     currentGen++
8:     Optimal= findBestFitnessPop(p)
9:     Emit(optimal)
10:    nextGeneration
11: end while
12: for unit : pop do
13:    parents=selection(pop)
14:    offsprings= crossover(parents)
15:    offspring= mutation (offspring)
16:    addoffSpringtonextGen (offspring)
17: end for
```

Algorithm 3 For reduce phase of GAs

```
1: for unit : pop  do
2:     checkoptimal(unit)
3: end for
```

5.2 Results

The proposed model first define the random population size and also define the number of mapper.After this every iteration of Genetic algorithms will used as the input of map phase. then calculating the fitness values of every iteration and find the efficient fitness. The selected fitness is crossover the generate the nw populations.

Genetic algorithm convergence with constant number of MapReduce tasks :

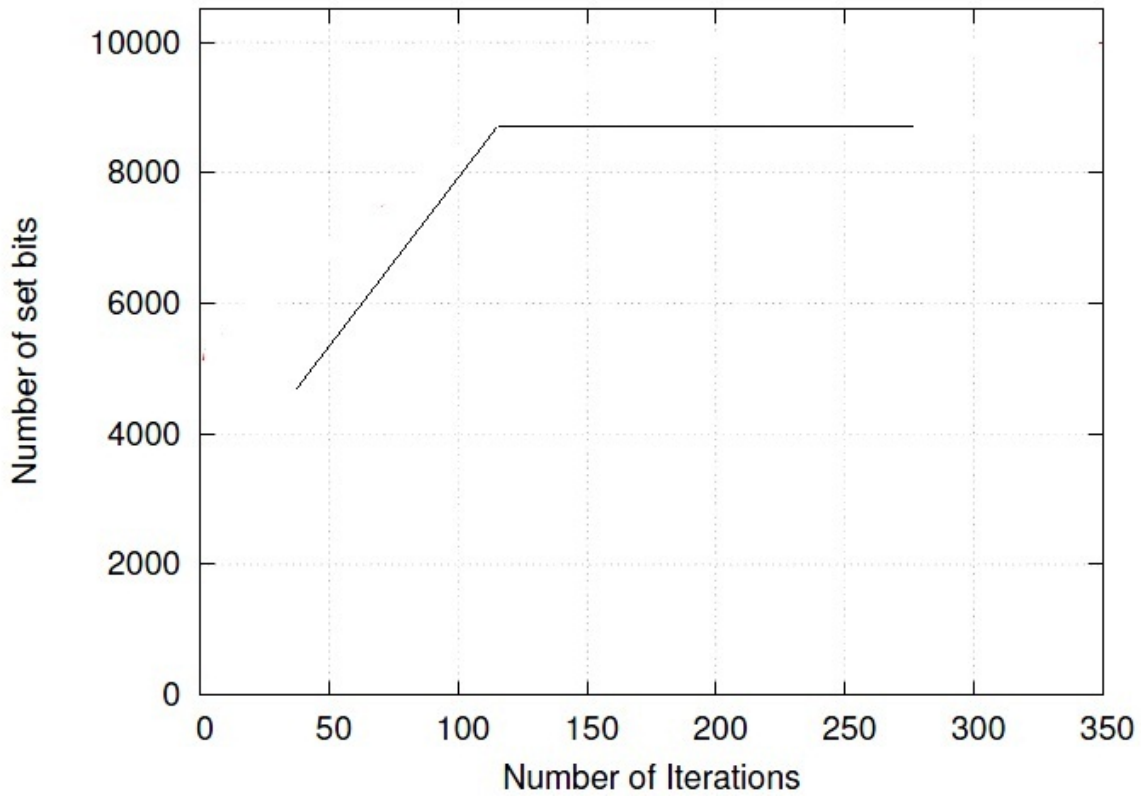

Figure 5.1: proposed Algorithms for MapReducing GAs.

GAs scalability with Fixed load per node in sets :

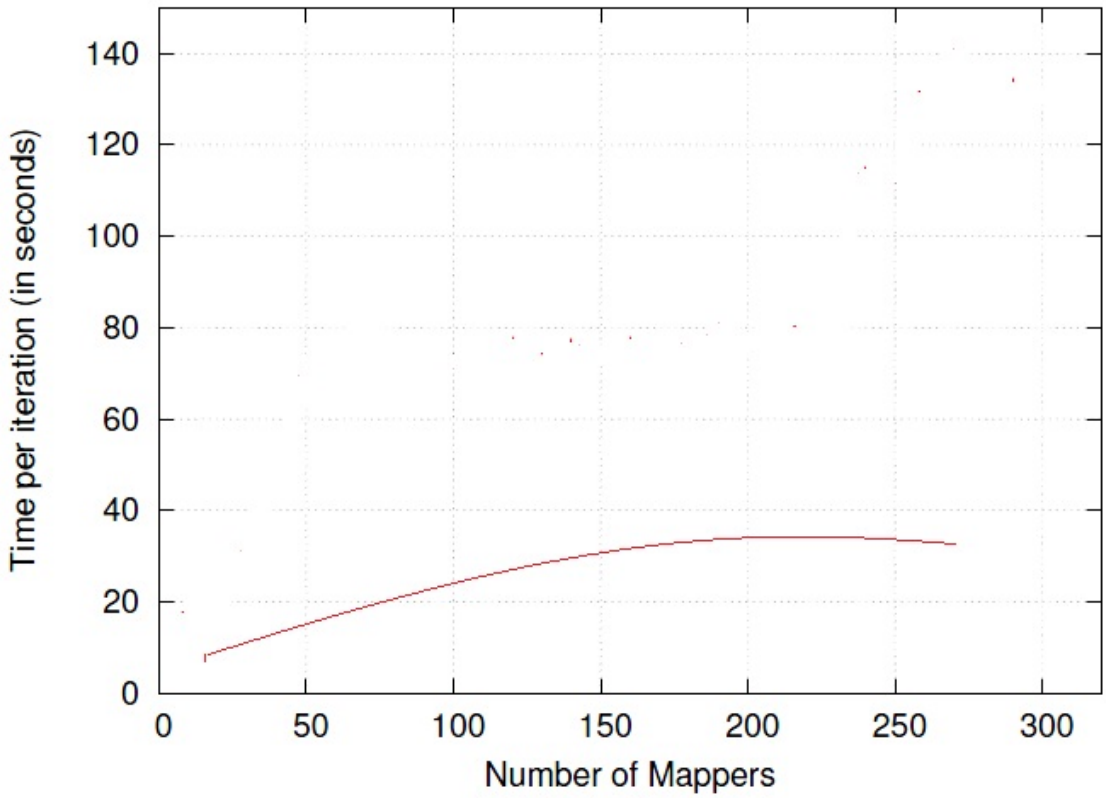

Figure 5.2: GAs scalability with Fixed load per node in sets.

5.3 Comparison the existing and proposed MapReduce Genetic Algorithms

For examining of genetic algorithm we are using One Max problem to compare with existing model [1]. In our implementation we are comparing Existing and proposed model by using one max problem. OneMax Problem (or Bit Counting) in this we maximize the number of one of a bit string.

We implemented OneMax problem on following configuration like 5 nodes cluster (i3 processor 2.6 GHz frequency and 4GB RAM, 300GB hard disk) with Linux O.S. and Hadoop distribution. Here performs two results of both model compared results:

1. Genetic algorithm convergence with constant number of MapReduce tasks,
2. GAs scalability with fixed load per node in sets

Genetic algorithm convergence with constant number of MapReduce tasks : Here we will using the best fitness values for testing both of model with fix the probability of crossover and mutation for fixed number of population size. In this model various genetic algorithm contents are following:

1. Prob(Crossover)= 0.6
2. Prob(Mutation) = 0.01
3. Total population = 10000

4.Mappers and Reducers = 15

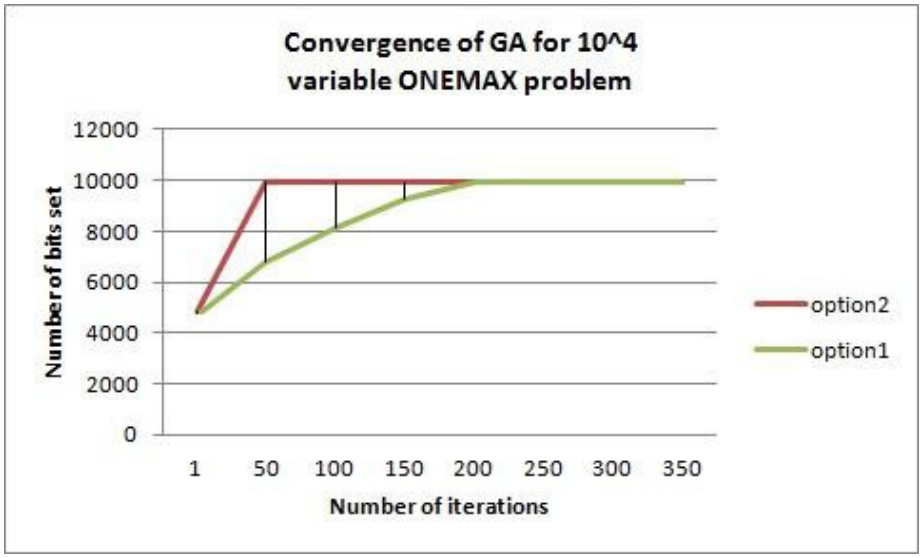

Figure 5.3: In this option 1 Every set of nodes have same populations; option 2 Every set of nodes have their own populations

Genetic algorithm convergence with constant number of MapReduce tasks : Here we will using the best fitness values for testing both of model with fix the probability of crossover and mutation for fixed number of population size.

In this model various genetic algorithm contents are following:

1. Prob(Crossover)= 0.6
2. Prob(Mutation) = 0.01
3. Total population = 6000

4.Variables in One Max problem = 10000

5.Iterations = 200

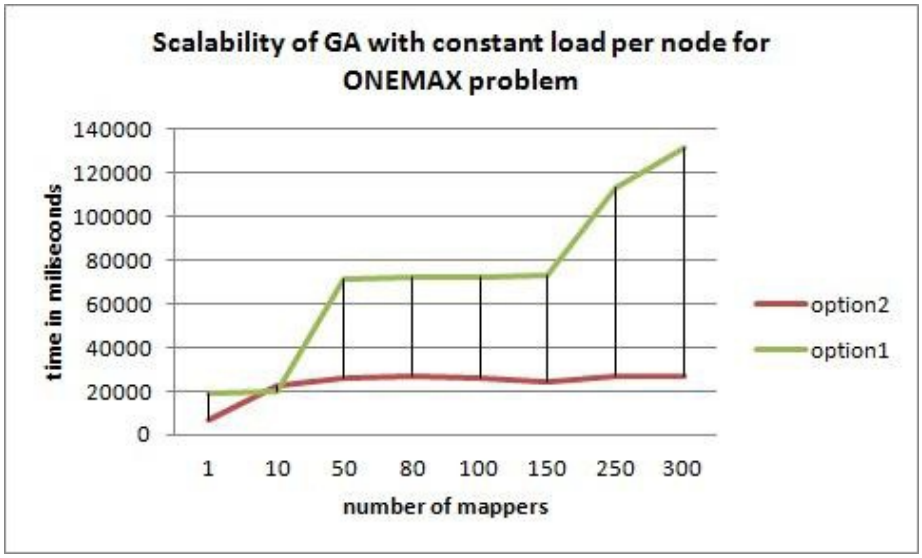

Figure 5.4: In this option 1 every set of node using same populations; option 2 Every set of nodes have their own populations

Chapter 6

CONCLUSION AND FUTURE WORK

In this thesis I show about the Genetic Algorithms using Data-Intensive Computing are possible. Here we shown the how step by step changes in case of simple selective recombination genetic algorithms and compact (estimated) distribution algorithms. Here we have shown step by step explanation that how the Genetic Algorithms, with different implementation form, will be interpret into Hadoop MapReduce framework. Here the results will give detailsas how Hadoop is best choice to impel genetic algorithm on large dataset problem and shown how the speedup will be increased using parallel computing. MapReduce is designed for large volume of data set. It is introduced for BigData Analysis and it is used a lots of algorithms like Breadth-First Search, Traveling Salesman problems, Finding Shortest Path problem etc. In this framework two key factor, Map and reducer. The Map which is parallely divided the data into many cluster and each cluster the data is form of key and value. The output of map phase data will goes into intermediate phase where data will be shuffling and sorting. Then using the partitioner for dividing the data parallelly in different cluster according to the user. In This thesis we also so that we compare our implementation with implementation presented

in existing model. These two implementation are compare with ONEMAX (bit counting) PROBLEM. The comparison criteria between two implementation are fitness convergences,stability with fixed number of node, quality of final solution, cloud resource utilization and algorithms scalability.

The future work of both models are used to solve many other problem like Traveling Salesman Problem. In proposed model some parts of MapReduce framework are not used in this problem like Combiner, So in future work we will show All phase of MapReduce framework.

Bibliography

[1] Cheng Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281, 2007.

[2] Abhishek Verma, Xavier Llora, David E Goldberg, and Roy H Campbell. Scaling genetic algorithms using mapreduce. In *Intelligent Systems Design and Applications, 2009. ISDA'09. Ninth International Conference on*, pages 13–18. IEEE, 2009.

[3] Michael Beynon, Chialin Chang, Umit Catalyurek, Tahsin Kurc, Alan Sussman, Henrique Andrade, Renato Ferreira, and Joel Saltz. Processing large-scale multi-dimensional data in parallel and distributed environments. *Parallel Computing*, 28(5):827–859, 2002.

[4] Shawn Bowers and Bertram Ludäscher. Actor-oriented design of scientific workflows. In *Conceptual Modeling–ER 2005*, pages 369–384. Springer, 2005.

[5] Dino Keco and Abdulhamit Subasi. Parallelization of genetic algorithms using hadoop map/reduce. *SouthEast Europe Journal of Soft Computing*, 1(2), 2012.

[6] Tekin Bicer, David Chiu, and Gagan Agrawal. A framework for data-intensive computing with cloud bursting. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 169–177. IEEE, 2011.

[7] Jeffrey Horn, Nicholas Nafpliotis, and David E Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 82–87. Ieee, 1994.

[8] Chrisila B Pettey, Michael R Leuze, and John J Grefenstette. Parallel genetic algorithm. In *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*, 1987.

[9] Xavier Llora. Data-intensive computing for competent genetic algorithms: a pilot study using meandre. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1387–1394. ACM, 2009.

[10] Chuck Lam. *Hadoop in action*. Manning Publications Co., 2010.

[11] Dhruba Borthakur. Hdfs architecture guide. *Hadoop Apache Project*, page 53, 2008.

[12] Chris A Mattmann, Daniel J Crichton, Nenad Medvidovic, and Steve Hughes. A software architecture-based framework for highly distributed and data intensive scientific applications. In *Proceedings of the 28th international conference on Software engineering*, pages 721–730. ACM, 2006.

[13] Jeffrey Shafer, Scott Rixner, and Alan L Cox. The hadoop distributed filesystem: Balancing portability and performance. In *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*, pages 122–133. IEEE, 2010.

[14] Grant Mackey, Saba Sehrish, and Jun Wang. Improving metadata management for small files in hdfs. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–4. IEEE, 2009.

[15] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D Stott Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040. ACM, 2007.

[16] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M Hellerstein, and Caleb Welton. Mad skills: new analysis practices for big data. *Proceedings of the VLDB Endowment*, 2(2):1481–1492, 2009.

[17] David E Goldberg. *Genetic algorithms*. Pearson Education India, 2006.

[18] Heinz Mühlenbein and Gerhard Paass. From recombination of genes to the estimation of distributions i. binary parameters. In *Parallel Problem Solving from NaturePPSN IV*, pages 178–187. Springer, 1996.

[19] David E Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, 3(5):493–530, 1989.

[20] Georges R Harik, Fernando G Lobo, and David E Goldberg. The compact genetic algorithm. *Evolutionary Computation, IEEE Transactions on*, 3(4):287–297, 1999.

[21] Shumeet Baluja. Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Technical report, DTIC Document, 1994.

[22] Martin Pelikan and Heinz Mühlenbein. The bivariate marginal distribution algorithm. In *Advances in Soft Computing*, pages 521–535. Springer, 1999.

[23] Erick Cantú-Paz. Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of heuristics*, 7(4):311–334, 2001.

[24] Philippe Giguere and David E Goldberg. Population sizing for optimum sampling with genetic algorithms: A case study of the onemax problem. *Genetic Programming*, 98:496–503, 1998.

[25] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters, osdi?04: Sixth symposium on operating system design and implementation, san francisco, ca, december, 2004. *S. Dill, R. Kumar, K. McCurley, S. Rajagopalan, D. Sivakumar, ad A. Tomkins, Self-similarity in the Web, Proc VLDB*, 2001.

[26] Yong Zhao, Michael Wilde, Ian Foster, Jens Voeckler, James Dobson, Eric Gilbert, Thomas Jordan, and Elizabeth Quigg. Virtual data grid middleware services for data-intensive science. *Concurrency and Computation: Practice and Experience*, 18(6):595–608, 2006.

[27] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: a survey. *AcM sIGMoD Record*, 40(4):11–20, 2012.

Dissemination

Conference

1. Rakesh Yadav and Dr. Ashok Kumar Turuk, Distribution of Genetic Algorithms Using Hadoop MapReduce, *The International Conference on Communication, Information & Computing (ICCICT-2015)*,Conference Proceeding ISBN 978-93-83006-07-6.